**MATLAB MINI PROJECT**

**OBJECTIVE:**

To perform time domain and frequency domain analysis of audio sample.

THEORY:

The MATLAB mini project aims to explore the time domain and frequency domain characteristics of an audio sample, offering valuable insights into its temporal and spectral properties. In the time domain, signals are analysed over a specific duration, unveiling information about amplitude variations and temporal patterns. MATLAB provides an ideal environment for this analysis through its extensive signal processing toolbox, enabling the extraction of features such as signal amplitude, duration, and temporal patterns.

On the other hand, the frequency domain analysis involves the transformation of the audio signal from the time domain to the frequency domain using techniques like the Fast Fourier Transform (FFT). This allows us to examine the spectral content of the audio signal, identifying prominent frequencies and their respective amplitudes. MATLAB's robust FFT implementation and visualization capabilities empower users to analyse the frequency composition of the audio sample effectively.

The project facilitates a comprehensive understanding of the audio signal's behaviour, bridging the gap between time and frequency representations. Through MATLAB's intuitive interface and powerful signal processing tools, users can gain valuable insights into the underlying characteristics of audio signals, making it an insightful exploration into the world of audio analysis.

MATLAB CODE:

1)Time domain analysis.

%%

clc;close all;clear all;

% Load the audio file

audioFile = '10secs.wav';

% [y, fs] = audioread('oneword.wav');

[y, Fs] = audioread(audioFile);

% Plot the waveform in the time domain

t = (0:length(y) - 1) / Fs;

figure;

subplot(2, 2, 1);

plot(t,y);

title('Original Audio Waveform');

xlabel('Time (s)');

ylabel('Amplitude');

% Objective 2: Effect of LPF filtering in Time Domain

% Design a low-pass filter using Butterworth

fc\_lp = 3000; % Cutoff frequency for low-pass filter in Hz

order = 8; % Filter order

[b\_lp, a\_lp] = butter(order, fc\_lp / (Fs / 2), 'low');

% Apply the low-pass filter to the audio using filter

y\_lp = filter(b\_lp, a\_lp, y);

% Plot the filtered waveform in the time domain

subplot(2, 2, 2);

plot( t,y\_lp);

title('Low-Pass Filtered Audio');

xlabel('Time (s)');

ylabel('Amplitude');

% Objective 3: Effect of HPF filtering in Time Domain

% Design a high-pass filter using Butterworth

fc\_hp = 1000; % Cutoff frequency for high-pass filter in Hz

[b\_hp, a\_hp] = butter(order, fc\_hp / (Fs / 2), 'high');

% Apply the high-pass filter to the audio using filter

y\_hp = filter(b\_hp, a\_hp, y);

% Plot the filtered waveform in the time domain

subplot(2, 2, 3);

plot(t, y\_hp);

title('High-Pass Filtered Audio');

xlabel('Time (s)');

ylabel('Amplitude');

% Objective 4: Segregate Voice and Un-voice parts based on frequency

% Design a low-pass filter for voice (below 6000 Hz)

fc\_voice = 6000; fc\_unvoiced = 50; % Cutoff frequency for voice in Hz

f\_pass=[fc\_unvoiced,fc\_voice];

[b\_voice, a\_voice] = butter(5, f\_pass / (Fs / 2), 'bandpass');

[b\_unvoiced, a\_unvoiced] = butter(order, f\_pass / (Fs / 2), 'stop');

% Apply the low-pass filter to the audio using filter to isolate voice

y\_voice = filter(b\_voice, a\_voice, y);

% Apply the high-pass filter to the audio using filter to isolate unvoiced

y\_unvoiced = filter(b\_unvoiced, a\_unvoiced, y);

% Plot the filtered voice and unvoiced signals in the time domain

subplot(2, 2, 4);

plot(t, (y\_voice));

title('Segregated Voiced Part');

xlabel('Time (s)');

ylabel('Amplitude');

player1=audioplayer(y,Fs);

play(player1);

pause(length(y)/Fs);

player2=audioplayer(y\_lp,Fs);

play(player2);

pause(length(y\_lp)/Fs);

player3=audioplayer(y\_hp,Fs);

play(player3);

pause(length(y\_hp)/Fs);

player4=audioplayer(y\_voice,Fs);

play(player4);

pause(length(y\_voice)/Fs);

player5=audioplayer(y\_unvoiced,Fs);

play(player5);

2)Frequency domain analysis.

% Load the audio file

[y, fs] = audioread('10secs.wav');

% [y, fs] = audioread('oneword.wav');

% Compute the FFT (Fast Fourier Transform) of the audio data

Y = fft(y);

% Compute the frequencies corresponding to the FFT result

frequencies = linspace(0, fs, length(Y));

% Take only the positive frequencies for the single-sided spectrum

positive\_frequencies = frequencies(1:length(frequencies)/2 + 1);

positive\_Y = 2.0/length(Y) \* abs(Y(1:length(Y)/2 + 1));

% Plot the single-sided spectrum

subplot(3,1,1);

plot(positive\_frequencies, (positive\_Y)); % Use log scale for better visualization

title('Single-Sided Spectrum');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

grid on;

% Apply Low Pass Filter (LPF)

cutoff\_frequency = 1000; % Adjust the cutoff frequency as needed

order = 4; % Adjust the filter order as needed

[b, a] = butter(order, cutoff\_frequency/(fs/2), 'low');

filtered\_data\_lpf = filter(b, a, y);

% Compute the FFT of the filtered data

filtered\_Y\_lpf = fft(filtered\_data\_lpf);

% Take only the positive frequencies for the single-sided spectrum

positive\_filtered\_Y\_lpf = 2.0/length(filtered\_Y\_lpf) \* abs(filtered\_Y\_lpf(1:length(filtered\_Y\_lpf)/2 + 1));

% Plot the single-sided spectrum of the filtered output

subplot(3,1,2);

plot(positive\_frequencies, (positive\_filtered\_Y\_lpf)); % Use log scale for better visualization

title('Single-Sided Spectrum after LPF');

xlabel('Frequency (Hz)');

ylabel('Amplitude ');

grid on;

% Compute the inverse FFT of the positive frequencies of the filtered signal

filtered\_signal\_time\_domain\_lpf = ifft(filtered\_Y\_lpf);

% High Pass Filter (HPF)

cutoff\_frequency\_hpf = 1000; % Adjust the cutoff frequency as needed

order = 4; % Adjust the filter order as needed

[b, a] = butter(order, cutoff\_frequency\_hpf/(fs/2), 'high');

filtered\_data\_hpf = filter(b, a, y);

% Compute the FFT of the filtered data

filtered\_Y\_hpf = fft(filtered\_data\_hpf);

% Take only the positive frequencies for the single-sided spectrum

positive\_filtered\_Y\_hpf = 2.0/length(filtered\_Y\_hpf) \* abs(filtered\_Y\_hpf(1:length(filtered\_Y\_hpf)/2 + 1));

% Plot the single-sided spectrum of the filtered output

subplot(3,1,3);

plot(positive\_frequencies, (positive\_filtered\_Y\_hpf)); % Use log scale for better visualization

title('Single-Sided Spectrum after HPF');

xlabel('Frequency (Hz)');

ylabel('Amplitude ');

grid on;

% Compute the inverse FFT of the positive frequencies of the filtered signal

filtered\_signal\_time\_domain\_hpf = ifft(filtered\_Y\_hpf);

% BandPass Filter

% Apply BandPass Filter

cutoff\_frequency = [100,6000]; % Adjust the cutoff frequency as needed

order = 4; % Adjust the filter order as needed

[b, a] = butter(order, cutoff\_frequency/(fs/2), 'bandpass');

filtered\_data\_voice = filter(b, a, y);

% Compute the FFT of the filtered data

filtered\_Y\_voice = fft(filtered\_data\_voice);

% Take only the positive frequencies for the single-sided spectrum

positive\_filtered\_Y\_voice = 2.0/length(filtered\_Y\_voice) \* abs(filtered\_Y\_voice(1:length(filtered\_Y\_voice)/2 + 1));

% Compute the inverse FFT of the positive frequencies of the filtered signal

filtered\_signal\_time\_domain\_voice = ifft(filtered\_Y\_voice);

% Plot the spectrum of the BandPass filtered output

figure;

subplot(2,1,1);

plot(positive\_frequencies, positive\_filtered\_Y\_voice);

title('Voice segregated Frequencies');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

% BandStop Filter

% Apply BandStop Filter

cutoff\_frequency = [100,6000]; % Adjust the cutoff frequency as needed

order = 4; % Adjust the filter order as needed

[b, a] = butter(order, cutoff\_frequency/(fs/2), 'stop');

filtered\_data\_noise = filter(b, a, y);

% Compute the FFT of the filtered data

filtered\_Y\_noise = fft(filtered\_data\_noise);

% Take only the positive frequencies for the single-sided spectrum

positive\_filtered\_Y\_noise = 2.0/length(filtered\_Y\_noise) \* abs(filtered\_Y\_noise(1:length(filtered\_Y\_noise)/2 + 1));

% Compute the inverse FFT of the positive frequencies of the filtered signal

filtered\_signal\_time\_domain\_noise = ifft(filtered\_Y\_noise);

% Plot the spectrum of the BandStop filtered output

subplot(2,1,2);

plot(positive\_frequencies, positive\_filtered\_Y\_noise);

title('Noise Part of Frequencies');

xlabel('Frequency (Hz)');

ylabel('Amplitude');

% Play the original audio

sound(y,fs);

pause(length(y)/fs);

% Play the filtered audio (LPF, HPF, BandPass, BandStop)

sound(filtered\_signal\_time\_domain\_lpf, fs);

pause(length(filtered\_signal\_time\_domain\_lpf)/fs);

sound(filtered\_signal\_time\_domain\_hpf, fs);

pause(length(filtered\_signal\_time\_domain\_hpf)/fs);

sound(filtered\_signal\_time\_domain\_voice, fs);

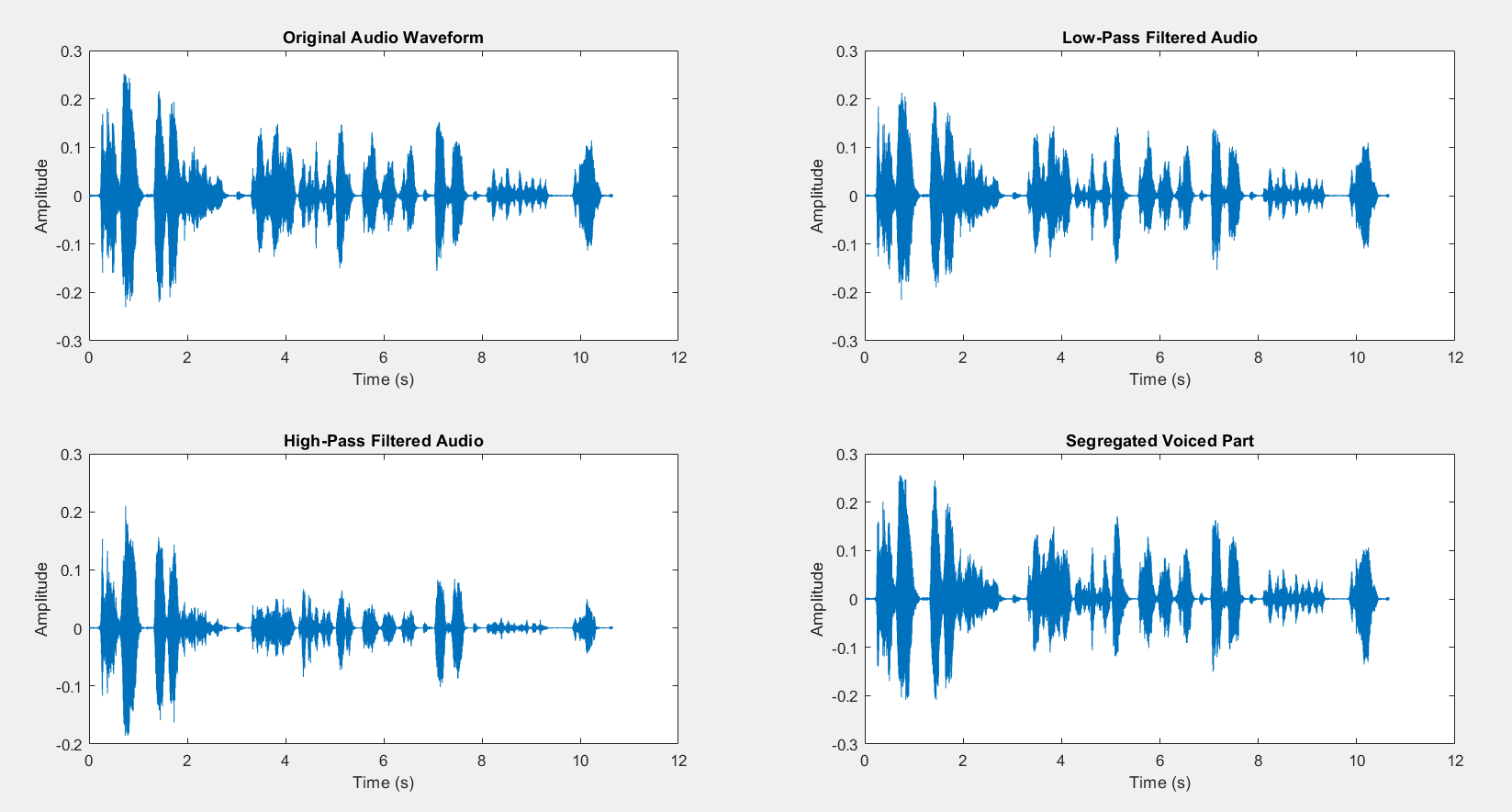
pause(length(filtered\_signal\_time\_domain\_voice)/fs);

sound(filtered\_signal\_time\_domain\_noise, fs);

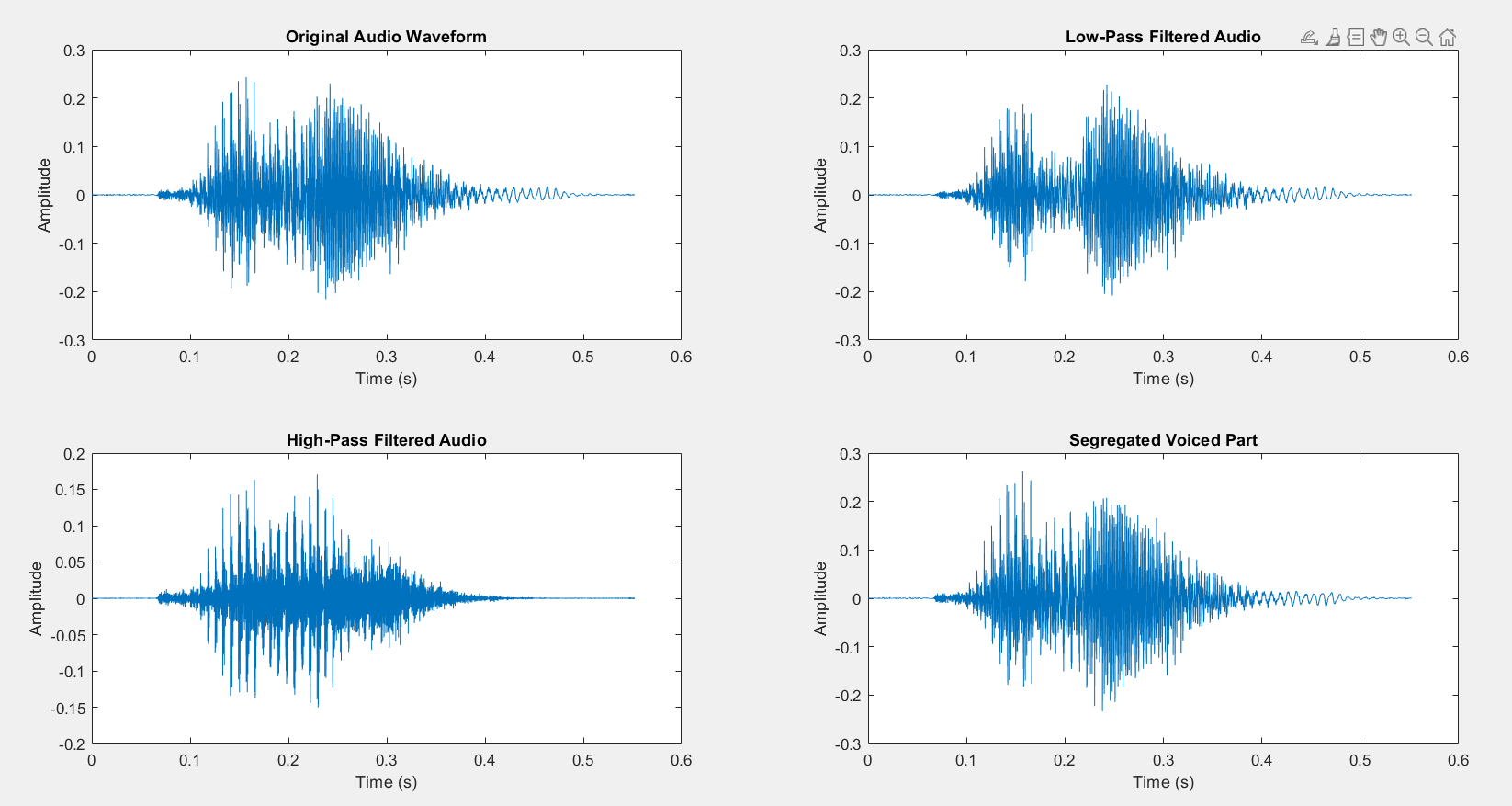
pause(length(filtered\_signal\_time\_domain\_noise)/fs);

MATLAB RESULTS:

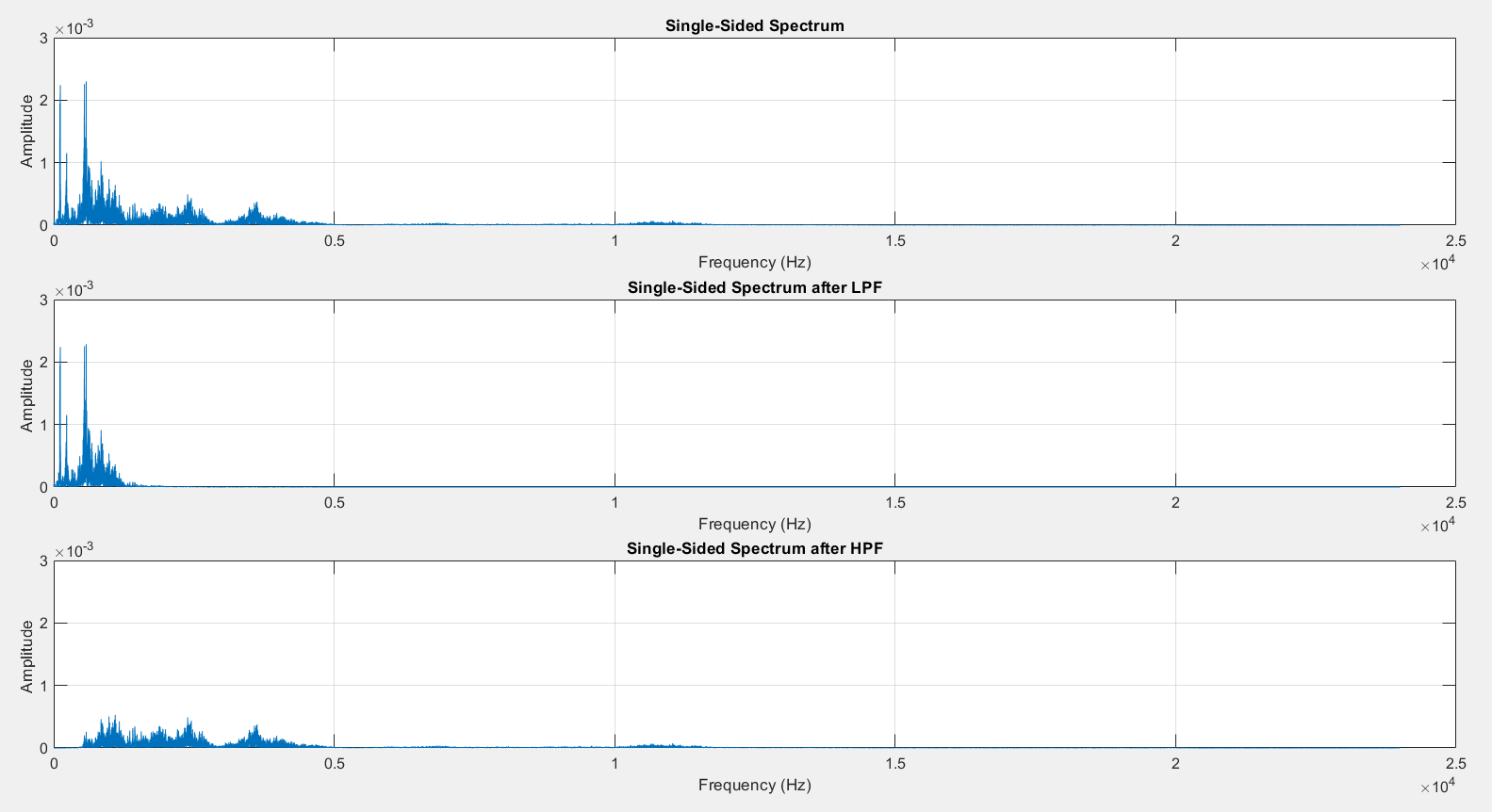
1)a) Time domain analysis for 10 seconds of voice.

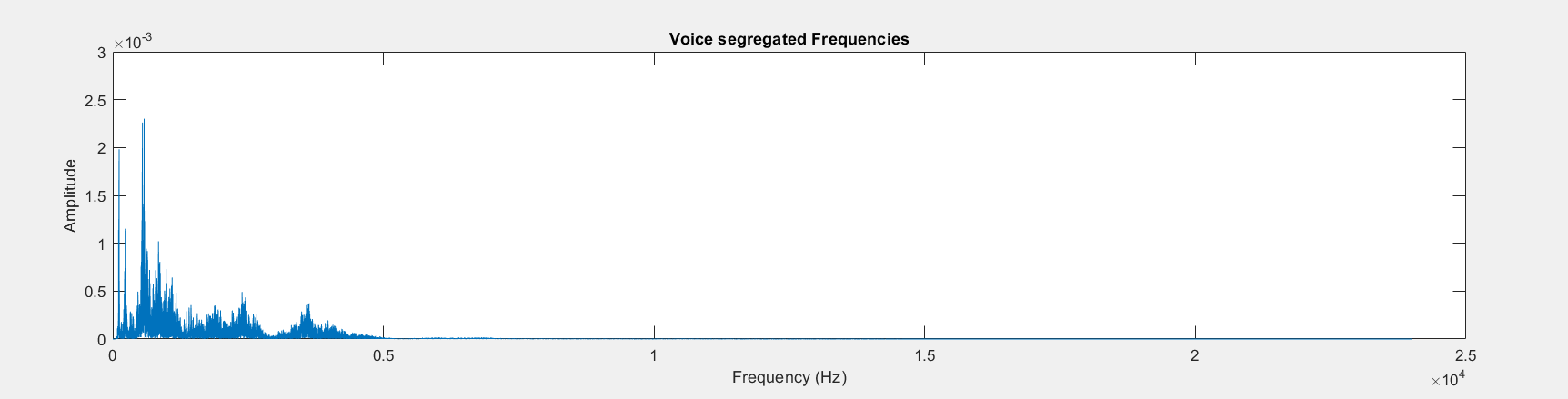


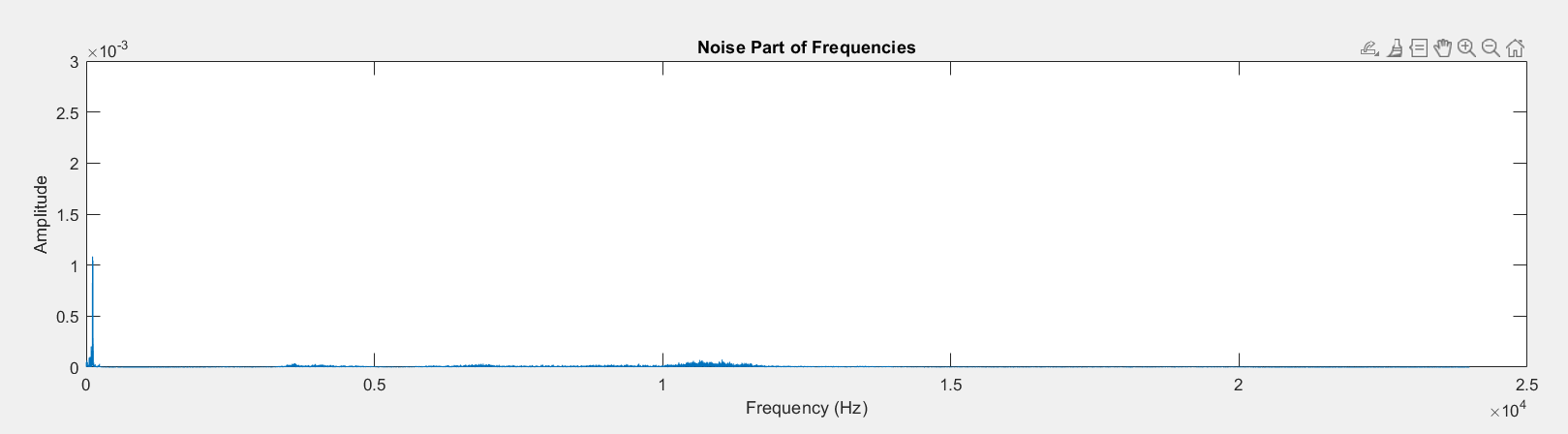
b) Time domain analysis for one word of voice.



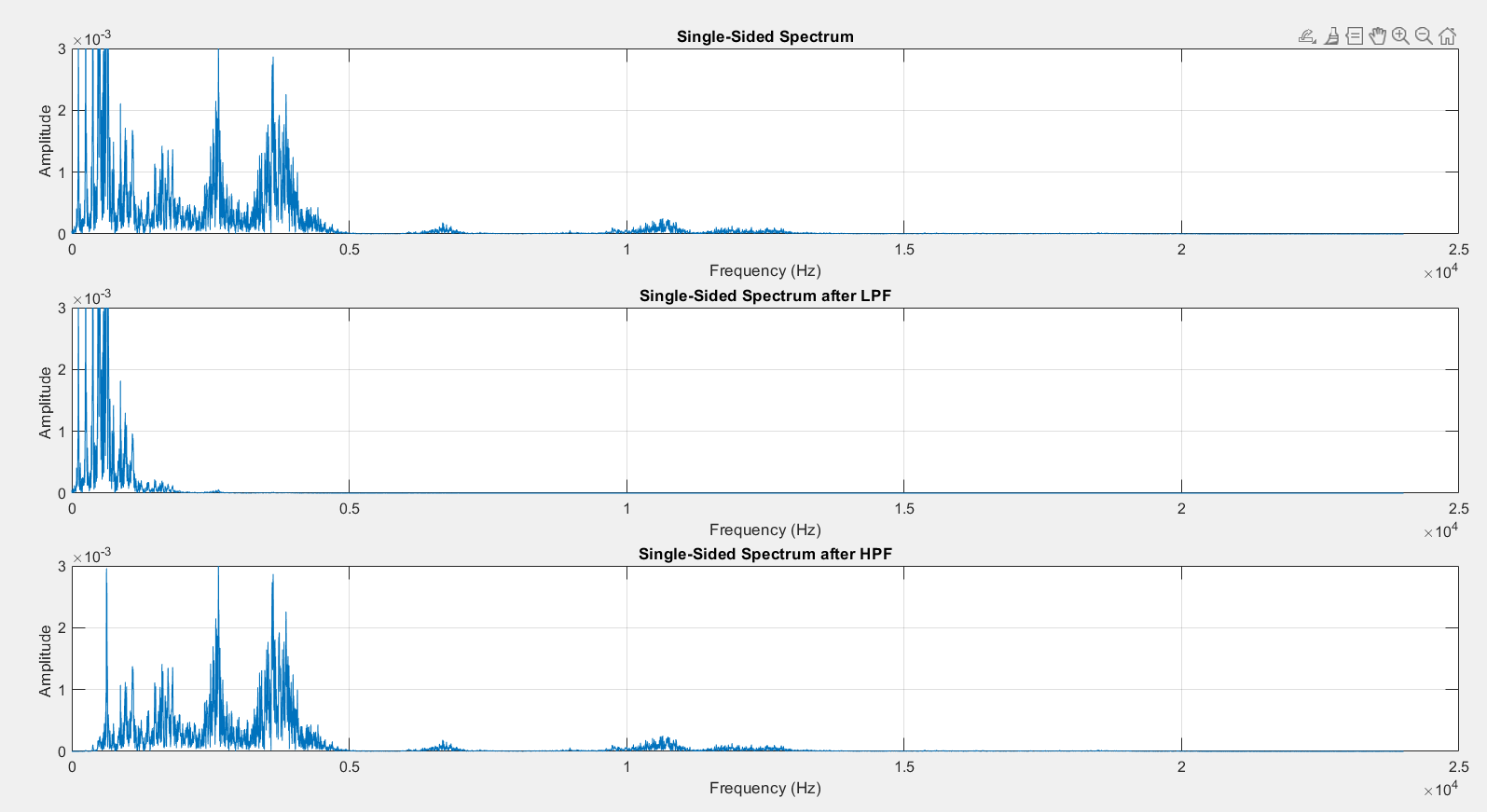
2)a) Frequency domain analysis of 10 secs of voice.

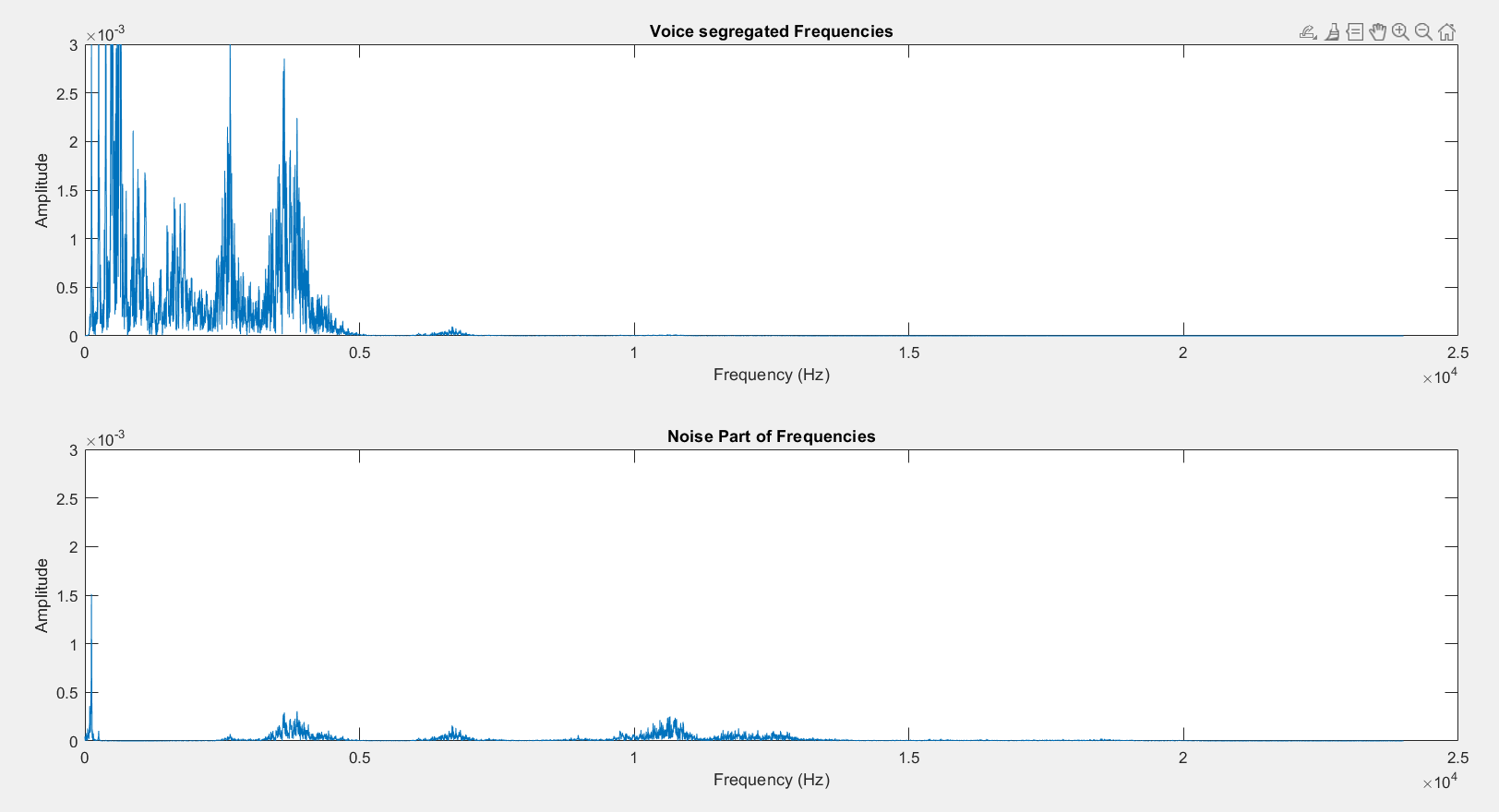






b) Frequency domain analysis of one word audio.





DISCUSSION ON RESULTS:

In the realm of audio signal processing, exploring the impact of high pass and low pass filters on a waveform extracted from a .wav file offers valuable insights into the manipulation of audio data. Initially, the audioread() function serves as the gateway to access the contents of the .wav file, providing a foundation for subsequent analyses. Upon acquiring the audio data, the application of high pass and low pass filters becomes pivotal. High pass filters attenuate frequencies below a specified cutoff, allowing higher frequencies to pass through, while low pass filters do the opposite. Analysing the effects of these filters in both the time domain and frequency domain unveils a transformation in the temporal and spectral characteristics of the audio signal. In the time domain, one observes alterations in the amplitude and shape of the waveform as a consequence of filtering. High pass filters accentuate the higher frequency components, emphasizing the nuances in the audio signal, whereas low pass filters preserve the lower frequency components, enhancing the bass and foundational elements. The frequency domain analysis, often achieved through the Fourier transform, visually captures the distribution of frequencies in the signal. Filtering accentuates or suppresses specific frequency ranges, shaping the spectral profile of the audio. This dual-domain exploration aids in comprehending how the filters sculpt the audio signal's overall structure. Further refinement involves the segregation of the voice and noise components, a critical task in audio processing. Employing sophisticated algorithms, MATLAB facilitates the extraction of distinct voice and noise segments. This segregation enhances the ability to manipulate and enhance the desired audio components, contributing to applications such as speech recognition, noise reduction, and audio restoration.

CONCLUSION:

In conclusion, the application of audioread, coupled with the judicious use of high pass and low pass filters in MATLAB, serves as a powerful avenue for delving into the intricacies of audio signal processing. The dynamic interplay between time and frequency domains, as revealed through waveform and spectral analyses, offers a nuanced understanding of how filters shape the characteristics of an audio signal. High pass and low pass filters emerge as versatile tools, influencing the temporal and spectral features of audio in distinctive ways. The ability to selectively enhance or suppress frequency components empowers practitioners to tailor audio output according to specific preferences or requirements. Whether accentuating the clarity of vocals with high pass filters or enriching the bass with low pass filters, these tools provide a means of sculpting audio content with precision. Furthermore, the segregation of voice and noise components opens doors to advanced audio processing applications. MATLAB's capabilities in isolating distinct segments contribute to advancements in fields such as speech recognition, noise reduction, and audio restoration. The synergistic integration of these techniques underscores the significance of MATLAB as a comprehensive platform for exploring, refining, and innovating within the realm of audio signal processing. In essence, this toolkit not only illuminates the transformative potential of filtering but also underscores MATLAB's role in shaping the future landscape of audio engineering and digital signal processing.

NAME-AKSHAR DASH

ROLL.NO-122EC0339